ESD-TR-68-350

Semiannual Technical Summary

Graphics

30 November 1968

Prepared for the Advanced Research Projects Agency
under Electronic Systems Division Contract AF 19(628)-5167 by

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Lexington, Massachusetts

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology; this work was supported by the U.S. Advanced Research Projects Agency of the Department of Defense under Air Force Contract AF 19(628)-5167 (ARPA Order 691).

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

GRAPHICS

SEMIANNUAL TECHNICAL SUMMARY REPORT
TO THE
ADVANCED RESEARCH PROJECTS AGENCY

1 JUNE – 30 NOVEMBER 1968

ISSUED 19 DECEMBER 1968

LEXINGTON MASSACHUSETTS

SUMMARY

A BCPL compiler is being transferred to the TX-2 from Project MAC. A microprogramming assembler has been written, and microcodes can be generated for a prototype processor under construction. The operation of the microcodes and the instruction sets they define can be simulated. An easily trainable public character recognition service has been made available to the TX-2 users community, and it can be called directly from LEAP programs.

The mask design facility has been improved and used to generate artwork for several test circuits. In addition, development of an IBM 360 capability has continued to the point of real use by Laboratory personnel. The AMBIT/G programming system has been refined in that drawn programs are themselves AMBIT/G data graphs. The Waveform Processing and RSP projects have been completed.

The TIC testing terminal is being augmented with a small stand-alone computer. The logic design and simulation programs have been further developed to permit the input of data, observation of values, and generation of test sequences for the circuit under design.

The conic display generator has had a major speed improvement, and storage scopes are being added to all TX-2 consoles.

Accepted for the Air Force
Franklin C. Hudson
Chief, Lincoln Laboratory Office

CONTENTS

	Summary	iii
	Glossary	vi
I.	LANGUAGES	1
	A. BCPL (Basic Combined Programming Language)	1
	B. Microprogramming	2
	C. Character Recognition Service in LEAP	5
II.	GRAPHICS AND APPLICATIONS	9
	A. Semiconductor Mask Design	9
	B. AMBIT/G	11
	C. Waveform Processing	17
	D. RSP (Re-entry Systems Program)	17
	E. Testing Terminal	18
	F. Computer-Aided Logic Design	18
	G. Conic Generator and Display Consoles	20

GLOSSARY

AMBIT/G	Graphical programming language for manipulation of directed graphs
APEX	TX-2 time-sharing executive
BCPL	Basic Combined Programming Language – an intermediate level language for computer programming
DOMEX	Display-Oriented Macro EXpander – an interactive control system for the microprocessor simulator
LDP	Link departure point
LEAP	Language for Expressing Associative Procedures – an ALGOL-like TX-2 programming language
LLL	Low-Level Logic
LSI	Large-scale integrated circuit technology
LX-1	A prototype microprocessor being constructed at Lincoln Laboratory
ML	A microprogramming assembly language
ROM	Read-only memory
RSP	Re-entry systems program
TIC	Testing integrated circuits
VITAL	A compiler-compiler system

GRAPHICS

I. LANGUAGES

A. BCPL (Basic Combined Programming Language)

BCPL is a simple recursive programming language, developed by M. Richards,* designed for compiler writing and system programming. To date, the uses of the language have included SNOBOL and MAD compilers, a QED editor, and a PAL interpreter.

The language has several interesting features. First, it does no type-checking at either compile or run time. In BCPL, all variables are considered bit patterns of a certain fixed length (determined by the machine on which the language is implemented), and the interpretation of those bit patterns is determined by the way in which they are used. Second, the language has a very comprehensive set of control commands, a few of which are:

if E do C
unless E do C
while E do C
until E do C

where E is an expression, and C is a command. The third interesting feature of the language is that it is fairly machine-independent, that is, the assumptions include: (1) memory must be a vector of fixed length words, each being addressable, and (2) it is possible to implement a pushdown stack on the machine.

The BCPL compiler is written in BCPL which makes the transfer of the compiler from one machine to another relatively easy. It has been implemented on CTSS and Multics at Project MAC, the System 360, and Atlas, among other machines.

BCPL on TX-2 will bridge the gap between LEAP and Mark 5. It is a high-level language yet its structure is simple enough that assembly language-like programming is possible, i.e., the BCPL programmer can manipulate pointers, push bits, etc., very easily and with complete confidence regarding the compiler's functions. Thus, fairly well optimized code can be created.

A second advantage of BCPL for TX-2 is that any programming done in BCPL will be more readily exportable to other computers. For example, had VITAL been implemented in BCPL, its use might have been more general. Conversely, with a BCPL compiler, we will be able to take advantage of other groups' work.

Third, the implementation of BCPL is such that interfacing the compiler with a TX-2 assembler will not be difficult.

In order to transfer the compiler from Project MAC's 7094 for implementation on TX-2, we had to solve three distinct problems:

(1) We rewrote the preprocessor (which accepts input text) and the code generator to produce a simple subset of Mark 5.

---

* M. Richards, "The BCPL Reference Manual," Project MAC Memo-M-352-1, M.I.T. (February 1968).

(2) We rewrote the BCPL library routines for TX-2.

(3) Since CTSS uses 7-track tape and TX-2 uses 9-track tape, we wrote a PL/I program for OS-360 to convert the CTSS tapes to TX-2 tapes.

All three steps in the transfer have been coded and partly checked out. The remaining areas of work are:

(1) To alter the code generator to produce binary, or, alternatively, to write a really simple assembler for the code produced by the compiler.

(2) To write a relocatable loader for the compiler programs, as well as to alter the compiler to produce relocatable code. This second problem has certain implications which have yet to be resolved. If a loader is produced exclusively for BCPL, all other potential compilers and assemblers will not be able to use it easily. This defeats one of the purposes of BCPL on TX-2, which is to improve communications between people and languages on TX-2. However, a full-blown loader is a large undertaking and it is unclear at this point what properties the loader should have.

(3) To add debugging aids to the BCPL system. This area has not yet been touched.

To date, two projects intend to use BCPL. The first project is to develop a new multiprocessing framework for TX-2; several parts of this system have already been coded in BCPL. The second project is to rewrite the code generator to produce code for the new testing machine and to write a real-time control and calculating system for the testing machine in BCPL.

B. Microprogramming

A programming package has been created on the TX-2 that allows a user to write and run microprograms written in the ML assembly language. Statements in this language correspond to control instructions for the LX-1 processor currently under construction. ML programs are free-format text files. The language allows symbolic labels, register names, literal and address constants, and is highly readable.

The ML programs are compiled by the ML assembler, implemented in VITAL into files of read-only memory (ROM) bit patterns. Side-by-side formatted listings are produced. The ROM files created drive the LX-1 simulator. Eventually, they will be read into the LX-1 control memory or be used to determine ROM bit patterns in subsequent LSI versions.

The LX-1 microprogram checkout and debugging package (LXSIM) provides a cycle-by-cycle simulation of the LX-1 prototype processor. The simulator was designed to allow maximum user flexibility. Snapshots of the current machine status are given via on-line display. Breakpoints can be put on microinstructions or on memory locations. Individual instructions can be altered or the whole program can be edited and recompiled. Memory locations can be examined and filled. Conditional execution or single stepping of the simulator is possible. New ROM or main memory files can be loaded at will. Different versions of the simulator, corresponding to various machine configurations, can be selected.

Since its purpose is to assist in debugging microprograms, the simulator is controlled interactively in a very flexible way. This flexibility is achieved because the simulator is designed to accept a large number of low-level text-encoded commands. If desired, the user can enter lists of these commands directly, via the outline keyboard, but to do so is quite tedious. These

basic commands, though quite flexible, are relatively primitive. For example, 72 characters must be typed to zero the 16 general registers. To combat inevitable user fatigue, a run-time command package has been written to allow the directed expansion of user-defined succinct commands into primitive commands.

User commands are defined by procedures written in the DOMEX (Display Oriented Macro EXpander) language, which has been based on the TRAC language.* Like TRAC, DOMEX is a language for text manipulation. Strings of characters may be named, parameter markers inserted, and strings called by name with argument lists. Character strings can be treated arbitrarily as executable procedures, names, or as text. Recursive function calls are also possible.

Unique to DOMEX is its ability to use the on-line display and Sylvania tablet. Light buttons can be defined and given procedural value. When a light button is pointed at, the procedure is executed, at times producing a new selection of light buttons. The procedure executed may also send strings of commands to the simulator. It can interrogate the status of the general registers in the simulated machine and use this information to control the generation of the command sequence sent. The procedure may also choose to demand characters from the tablet. The character recognizer will be called and its output made available to the procedure.

Part of the flexibility of DOMEX commands is their run-time definition. When the simulator package is first called, the only defined DOMEX command is one that will read and execute text files. An initialization text file might contain a DOMEX procedure which, when executed, defines other commands. Different users may have different initialization files. The user is free to drop or edit old definitions or to create new ones. DOMEX procedures can define new ones or be self-modifying. New initialization files can be written out at any time.

Typical simulator control procedures that can be defined might include:

(1) Single step until a selected register contains a given value

(2) Load one register from another

(3) Remember the machine status so it can be restored later – even at a different session

(4) Xerox read-only memory in symbolic format.

(5) Trace a register, typing the value and the microprogram instruction counter every time it is changed

(6) Edit, recompile, and reload the ROM file

(7) Write out a text file which, when read in, redefines all strings currently in use

(8) Trace an instruction, printing out selected register values every time it is executed.

The selection and form are, of course, up to the user. The command package can be tailored as desired.

Figure 1 shows the active display when LXSIM is first executed. No ROM or memory files have been brought in. The default names, R0, R1, . . . , R17 have been assigned to the 16 general registers. Input is expected from the Lincoln Writer.

Figure 2 shows the display after reading in an initialization file and having run the simulator. ROM file TEST · 1 has been loaded and MLSIM is the version of the simulator. R2 has

* C. N. Mooers, "TRAC, A Procedure-Describing Language for the Reactive Typewriter," CACM 9, No. 3, 215-219 (March 1966).

-20-8492

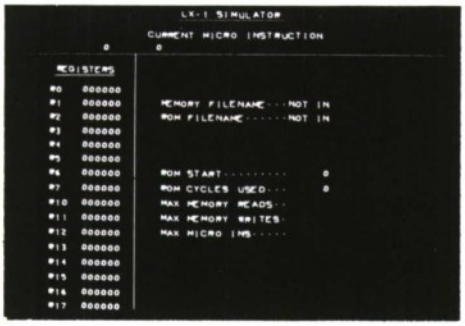


Fig. 1. Uninitialized simulatar.

-20-8493

Fig. 2. Simulator braken during executian.

-20-8494

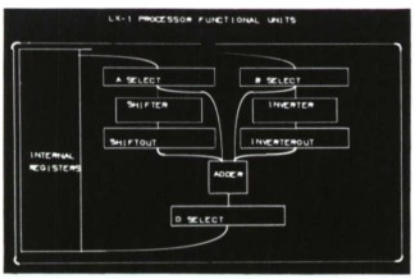


Fig. 3. Organizatian of LX-1 processar.

-20-8495

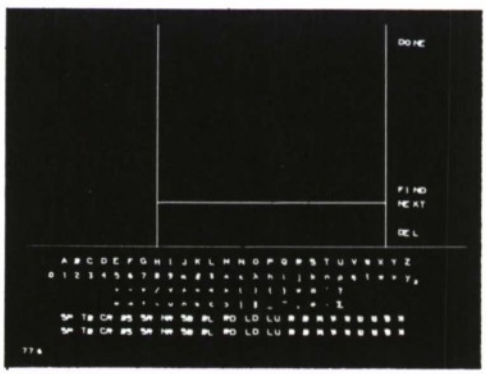


Fig. 4. Basic display af recognitian training pragram.

been given the new name "CTRL." The simulation has stopped at ROM location 10 because the simulation will exceed the maximum number of microinstructions allowed in one step. Location zero of main memory contains a zero. Three light buttons appear because input is expected from the TABLET.

A display which illustrates the functional organization of the LX-1 processor is shown in Fig. 3. Each of the three busses can be connected to one of the 16 internal registers. The contents of the A-select register are connected to a shifter, and are shifted or relocated up to 16 positions. The B-select contents can be optionally complemented. The outputs are fed into a logic box which does one of eight functions: add, logical and, or, exclusive or, A-only, B-only, read, or read/write to scratch pad memory. The results replace the D-select contents or are forgotten if D-select is zero. Register 1 contains the address of the current microinstruction and six control flags. The condition flags may be used for shift-in or carry-in operations, or to perform two- or four-way branches in the microcode. Other dedicated registers are for memory address and memory buffer, and for memory and I/O control. This display shows the user a picture of simulated machine operation.

C. Character Recognition Service in LEAP

A recognition capability for symbols drawn on the Sylvania Tablet has been made easily accessible to TX-2 programmers. A simple way of training the recognizer for individual symbol sets has been provided, and a set of LEAP language forms for calling the recognizer and trainer have been created. Because of these conveniences, the recognition facility has been used extensively in a number of applications, and the results have been very satisfactory and enlightening.

The user calls the training program with the name of the symbol set which he wishes to define or augment. The trainer initially displays the picture shown in Fig. 4. The lower half of the screen contains an underlined touch-sensitive target for each character or character string which is associated with some symbol in the symbol set. In addition, the rest of the characters in the TX-2 character set are displayed but not underlined.

The user initially draws a symbol in the working area at the top center of the screen as shown in Fig. 5. When the symbol is complete (i. e., when a new stroke is not started within a certain time interval after termination of the previous stroke), the trainer displays an enlarged smoothed copy of the drawn symbol and then analyzes the symbol. There are three possible results:

(1) If the symbol is not recognized at all, either because the drawn symbol is not close to any entry in the symbol set or because it was equally close to two or more entries, "??" is displayed in the rectangle just below the working area, as shown in Fig. 6. (The numbers in these pictures are a representation of the information which the recognizer extracts from the drawn symbol. They are generally ignored by users, but, if two symbols become confused, the user can interpret this information to help diagnose the conflict.)

(2) If the symbol is closer to one entry in the symbol set than to any other, but not an exact match to any entry, the string associated with that entry is displayed followed by a "?" as shown in Fig. 7.

(3) If the symbol exactly matches one entry in the symbol set, the associated string is displayed followed by a "." as shown in Fig. 8.

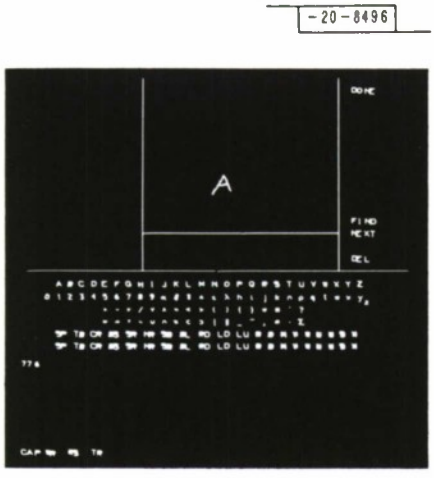


Fig.5. Drown symbol hos been entered.

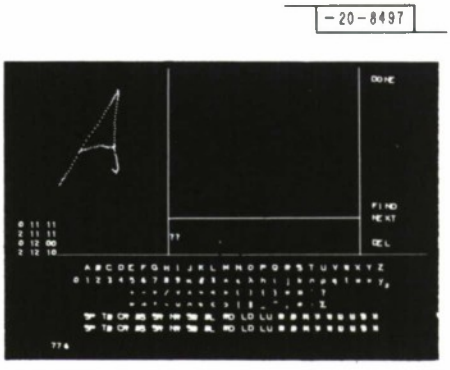


Fig.6. Drawn symbal is nat recagnized.

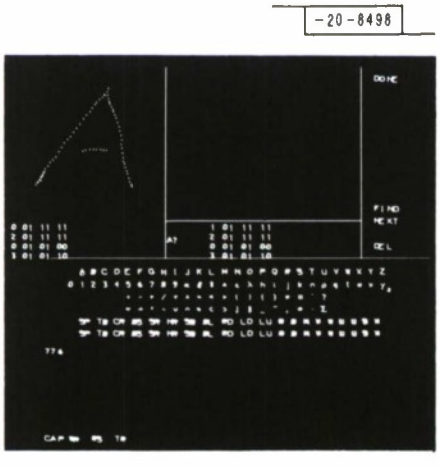


Fig. 7. Symbal has been recagnized as potentiol "A."

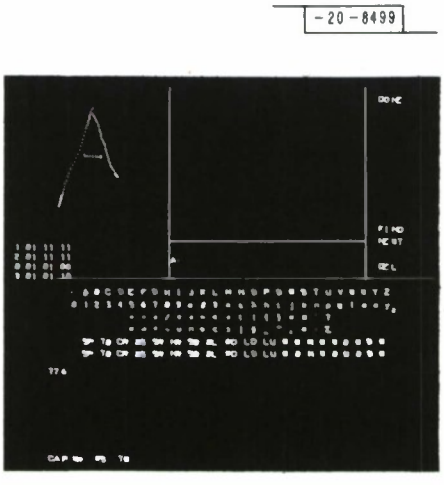


Fig.8. Symbol has definitely been recognized as an "A."

The user may then define or redefine the symbol he has just drawn by either touching the character string on the lower part of the screen which he wishes to associate with the symbol, or by typing a new string (which will appear on the bottom of the screen from that point on). In case (2) above, he may alternatively touch the string in the rectangle below the working area instead of the corresponding string on the lower part of the screen.

In case (2) or (3) above, the user may also delete the matched entry from the symbol set by touching the DELETE target. After deletion of a symbol entry, the symbol set is re-examined for other close matches just as if the symbol had been redrawn.

At any point, the user may draw a new symbol in the working area to repeat the whole process, or touch the DONE target to terminate the training session.

The user can also step through the entries in the symbol set associated with a given character string by touching the FIND target and then touching a string on the lower half of the screen. The trainer then acts as if the user has just drawn a symbol which exactly matches the chronologically last entry in the symbol set which is associated with the string. Touching the NEXT target will step to the preceding entry for the string or indicate that there are no more. This facility is used for selective deletion of unwanted entries, for redefining the strings associated with symbols, or for examining the numeric representation of the information extracted by the recognizer in analyzing the symbols corresponding to a given definition.

Users have found this training facility very convenient, both for defining new symbols as they are needed and for improving the frequency of successful recognition of a given symbol when it is discovered that the initial training was insufficient. In fact, there is a facility in LEAP for calling the trainer from a LEAP program without disturbing the program. Most users define a symbol which will, when drawn and recognized by the program, call the trainer and allow them to augment the recognizer being used.

An improvement to the trainer would be a facility for merging two symbol sets into one. For example, a user might have a recognizer for alphanumeric characters and another for circuit symbols, and may wish to merge them in order to be able to write textual information of the circuit diagrams which he draws. The problem with merging symbol sets is the resolution of conflicts. This could presumably be handled by simply asking the user to redefine or redraw one of the symbols in each conflicting pair.

LEAP has language forms for dealing with nontypewritten interactive input as described in the previous Semiannual Technical Summary.* Through these, the user can request the time-sharing and LEAP systems to automatically display the pen track during drawing and wake up the user's program when a pause in drawing occurs. At this point, the program can examine the pen track itself or execute the statement "RECOGNIZE" which will analyze the pen track as one of the symbols in the symbol set which has previously been specified by the user, or report that it does not recognize the drawn symbol. Reserved variables are set by RECOGNIZE for the text associated with the symbol and for the position and dimensions of the symbol.

The following is a skeleton LEAP program which indicates what the user must do in order to use the recognition facility in his system. Note that the program is independent of the actual

* Semiannual Technical Summary to the Advanced Research Projects Agency on Graphics, Lincoln Laboratory, M.I.T. (31 May 1968), DDC 671125.

appearance of the drawn symbol; the user concocts the symbol he wishes to use for a given function and simply associates it with the appropriate text during training.

```
        START {FILE R};                 Declare an input parameter R of data
                                        type "file name".
        R → RECOGNIZER;                 Store the file name into the reserved
                                        variable RECOGNIZER to establish the
                                        file as the symbol set to be used by the
                                        character recognition facility.
        ACTIVATE INKING;                Request the system to display and
                                        buffer the pen track; other devices
                                        such as button pushes, light target hits,
                                        etc., may also be activated.
NEXT:   GETNEXTINPUT;                   Pop an event from the input queue
                                        (pausing until an event occurs if the
                                        queue is empty) and set CAUSE to the
                                        identification number for that event.
        SWITCH VIA CAUSE                Branch on the event number to the
        TO···,TARGET,···,BUTTON,        appropriate processing section for the
        ···,NEWSYM,···;                 event.
NEWSYM: RECOGNIZE;                      Analyze the pen track as a symbol.
                                        The symbol set named in "RECOGNIZER"
                                        is used. The reserved variable SYMBOL
                                        is set to the associated text string;
                                        XCEN, YCEN, WIDTH, HEIGHT indicate
                                        the center and dimensions of the drawn
                                        symbol.
        IF SYMBOL = '' THEN···          Symbol not recognized, i.e., equal to
                                        null string.
        IF YCEN < -0.9 THEN···          Symbol drawn on lower edge of screen.
        IF SYMBOL = 'AAB' THEN···       Symbol for "AAB" recognized.
TARGET: ···                             Process target hit.
BUTTON: ···                             Process button push.
        FINISH
```

One common use of the recognition facility is to eliminate keyboards, pushbuttons, and light target arrays for specifying control functions. Keyboards and pushbutton boxes tend to get in the user's way at the console, and he must turn his attention away from the screen in order to hit the right key. The keys are also limited in number; in order to add a new function once all keys have been assigned, it is necessary either to use a combination of keys or install a new key. Light targets are more flexible, but they tend to clutter the picture, increase flicker rate, and reduce the area available for displaying pictures or text. We have found that drawing symbols to initiate a function is more convenient than either of the above techniques. These symbols might be either single characters or noncharacter symbols at the individual user's option; he may give whatever value he wishes to any symbol he can draw. Conflicts in the meaning of symbols (e.g., if a "Q" is sometimes a character in a label and sometimes a command to "quit") can be resolved by changing to "label" mode or by requiring control commands to be drawn only in one corner of the tablet. This latter technique also reduces the possibility of accidentally initiating drastic actions such as clearing the screen by unintentionally drawing the wrong character.

An extension of the use of symbols for control is to use the location of the drawn symbol as well as its value. For example, a "C" drawn at any point on the screen might be a command to translate the picture so that it is centered about the point at which the symbol was drawn; an "X" drawn on a picture component might mean to delete that component; and a "T" might mean that a transistor picture is to be displayed where the symbol was located. The size of the symbol might also convey information; for example, the width of the symbol for "resistor" (as specified in WIDTH upon return from RECOGNIZE) might indicate the length of the component leads.

This latter class of applications overlaps into the area of picture drawing, where additional capabilities would be useful. For example, rather than using the length of the "resistor" symbol to specify the length of the entire component, it would be desirable for the recognizer to return information which could allow the programmer to easily detect the "real" center of the symbol (i. e., where the "wiggle" is located) so that different size leads from either side of the resistor element could be specified. Another example which cannot be handled well by our present system is the recognition of "arrows" independent of orientation. What seems to be needed here is to make the features extracted by the recognizer available to the user program for further analysis. However, it is not clear what makes a good general set of features, which can be easily interpreted by the programmer.

Even without such a facility, however, we have applications with picture drawing features (particularly the mask layout program described later) which use the symbol recognition facility as the major interface with the user. We have found that when programmers are provided with an easily accessible tool, they find ways to adapt it to their particular application.

II. GRAPHICS AND APPLICATIONS

A. Semiconductor Mask Design

The typewritten pattern generator program on TX-2 has been used to design a chip for testing fuses to be used in loading ROM's and for a chip to be used in heat dissipation studies. The graphical version has been used to lay out masks for a chip which contains gate-chains for measuring stage delays at two power levels – 5 and 10 mW per gate.

This new version of the sketching program takes advantage of recent additions to the time-sharing system APEX and the programming language LEAP. The program is now organized so that new component definitions and new design rules can easily be incorporated, and it includes more powerful picture manipulation facilities. The newly used APEX facilities include the interrupt handler, the feedback display of the pen's current position, and the pen track display (ordered accumulation and display of points where the pen has been during the current activity period). The newly used LEAP facilities include the easy linkage of a LEAP program to public programs, such as the keyboard scope editor, the tablet scope editor, the character recognizer, and the character recognizer's trainer. Also, the LEAP facility for merging two LEAP structures can be used to combine two sets of circuit patterns.

The new version of the program is organized so that component pattern definitions and design rules, embedded in the program, can easily be added to, subtracted from, and changed. This design goal was successfully tested when the component pattern definitions and design rules were changed from bipolar to MOS technology in a period of $2\frac{1}{2}$ man hours. These two technologies are dissimilar in their design rules and component pattern definitions.

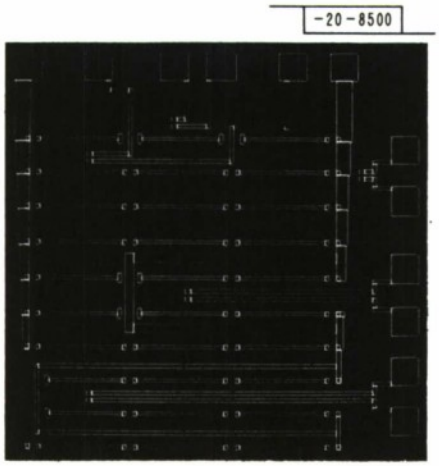


Fig. 9(a). A portion (scope view) of first-level metal pattern for resistor chip (heat test).

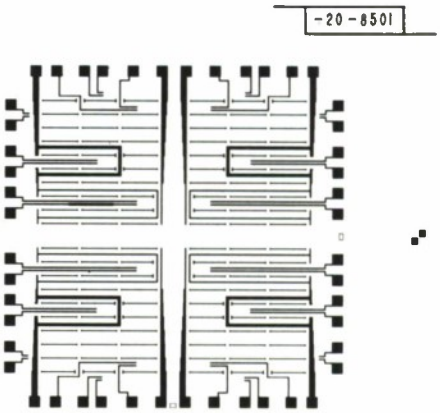


Fig. 9(b). Metol pattern (Master Reticle) for resistor chip.

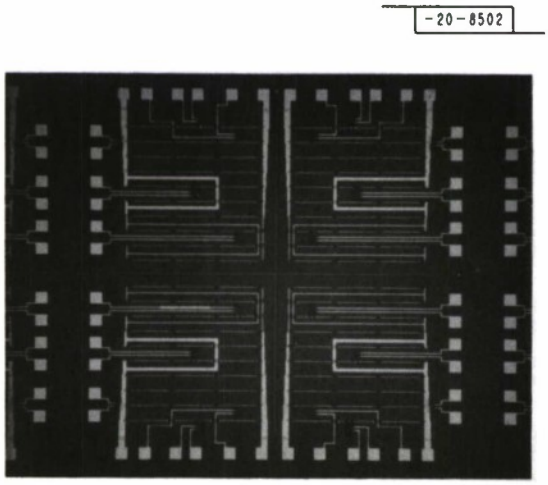


Fig. 9(c). Microphotograph of resistor chip.

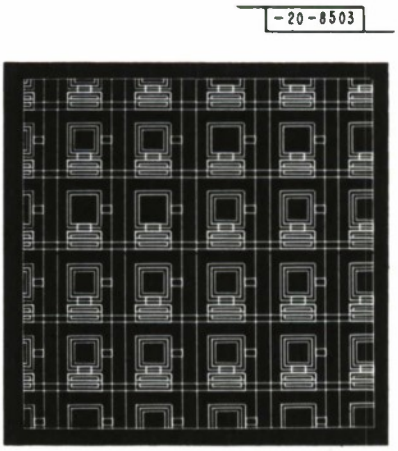


Fig. 9(d). Composite view of new version of ROM.

More powerful pattern manipulating functions are included in this version of the program. These new facilities provide for windowing of large pieces of artwork, creating second- and third-level metal patterns and vias, creating higher order pattern definitions (groupings of previously defined patterns), and merging two sets of artwork patterns. The program also has the ability to use the storage tube as well as the dynamic display available on TX-2's consoles. Large scale (30 × 30 inch) accurate hardcopy of any pattern is also available from a plotter on Lincoln Laboratory's IBM 360. All these facilities were necessary to accommodate the more complex circuits that have been designed.

The written scheme for generating patterns has been implemented in Fortran IV on the Laboratory's IBM 360 equipment. Patterns are described in Fortran IV and viewed on an IBM 2250 CRT or on hardcopy. The pattern information is punched into paper tapes which are used to actually generate the patterns for use in the microelectronic fabrication. The IBM 360 version of the written pattern generation scheme is now being used on a production basis, by at least one other group within the Laboratory, to generate patterns.

Recently, efforts have been made to extend the capability of this production facility. A scope-driven text editor has been written to facilitate the composition and editing of the Fortran IV subroutines that specify the patterns. This editor makes use of the primitive text-editing features built into the 2250 Graphics Terminal, such as cursor manipulation and character replacement. Other functions such as page-turning, line insertion, and deletion are requested via the programmed function keyboard which causes a CPU interruption. The program works quite well under the time-sharing system as the scheduling algorithm seems to favor users with I/O devices that request service frequently and asynchronously.

In addition, the Laboratory users of the system have now modified the original IBM 360 pattern description package to allow the user to enter mask data from his console and see it displayed immediately on the CRT. This is a feature which helps to decrease the time required to produce a correct pattern. Once the pattern has been adjusted to the user's satisfaction, he may then request hardcopy and/or paper tape output.

Figures 9(a) through (d) and 10(a) through (d) show some of the stages in the design and realization of the resistor heat chip, the gate chain, and the revised ROM.

B. AMBIT/G

The AMBIT/G graphical programming language facility has been extended and modified in two major and many minor respects. The first major change involves a simple but far-reaching alteration in the specification of the AMBIT/G language itself. Flow of control in AMBIT/G has in the past been indicated by associating with each program statement success and fail labels which determine the statement to be executed next. Now, program statements are considered as special cases of subprograms which have only two exits; subprograms in general can have any number of exits. These changes allow the flow of control in an AMBIT/G program to be graphically specified. To define a subprogram, the programmer first enters subprogram-definition mode. Light buttons appear indicating the names of all subprograms and program statements currently defined – that is, all things which can be called as subroutines. The programmer specifies the name of the subprogram he wishes to define and an entry point appears (see Fig. 11). He then hits light buttons and using an "X" places the subroutine calls on the screen.

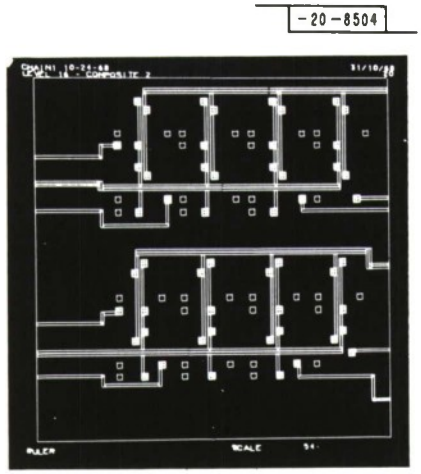


Fig. 10(a). Composite view of second-level metal and first-level pods of gate chains.

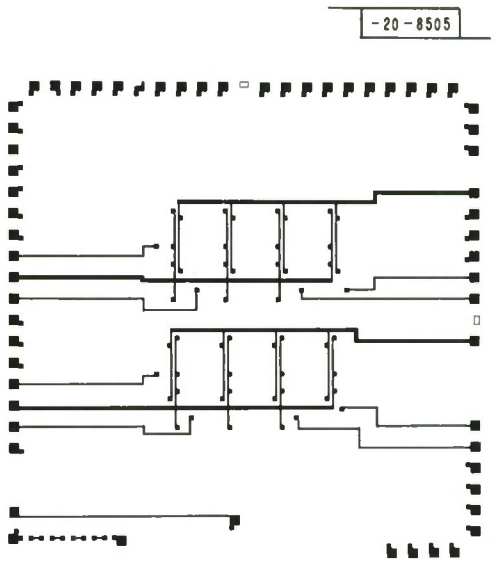


Fig. 10(b). Second-level metal pattern (Master Reticle) for gate chain.

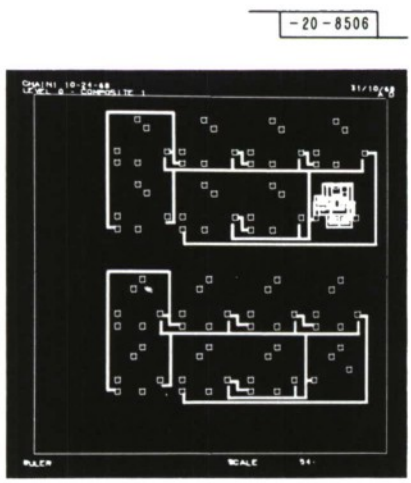


Fig. 10(c). Composite view of first-level metal pattern, including detailed specification of one gate, far gate chain.

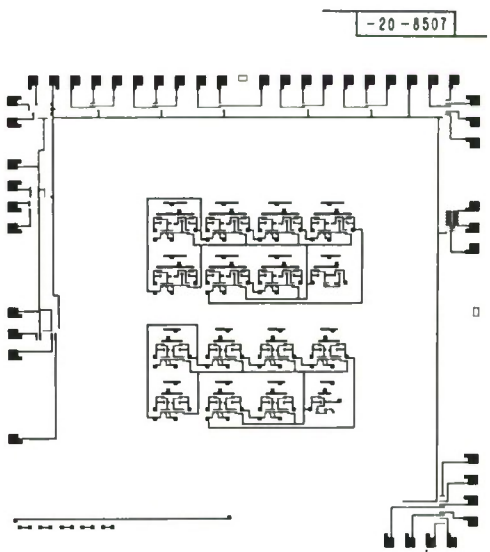


Fig. 10(d). First-level metal pattern (Master Reticle) for gate chain.

The exit points appear in the calls and are joined by arrows, thus indicating flow of control (see Fig. 12). Exit points can be added and connected to output control lines. The completed definition may be filed away for future reference. Subsequently, its name will appear as a light button, allowing it to be called.

A defined subprogram may be edited. Since during editing the subprogram is defined, it can be used in defining itself. This allows one to call a subprogram as a subroutine from within itself, making recursion possible (see Fig. 13).

The main subprogram is defined this same way (see Fig. 14), but exit points are not allowed. Also, its name will never appear as a light button, ensuring that it cannot be called from another subprogram. The special program statement named STOP is known to the system and will appear whenever one is in subprogram mode.

The interpreter begins at the entry point of the main subprogram and follows the specified control flow until it reaches a call to the STOP program statement. As not all exit points of all calls need be connected by arrows (presumably certain exits will never be used), it is possible to have the flow unspecified due to a programming error. This information is reported when encountered at run time.

Along with the neatness of specifying program flow graphically comes another (as yet, unexploited) benefit. The subprogram definitions can easily be treated as just another part of the AMBIT/G data base. If this were done, self-modifying AMBIT/G programs could be written. The details and implications of this have yet to be explored.

The second major change has resulted from an observation made by users of the AMBIT/G language who have discovered that they were writing many near-duplicate statements due to the inability to talk about classes of shapes. Hence, the facility to specify that a certain shape will stand for any of a class of shapes and tokens has been added to the language and the system. The class shape is designated by a "C" in the center of it; tokens of a class, of course, are nonexistent. A class definition mode is provided in the system in which instances of those shapes and tokens which are to be members of the class are placed with the class shape. Then, each of the link departure points (LDP's) of the class is connected to an LDP of each of the class members (see Fig. 15). This correspondence between class LDP's and class member LDP's is necessary so that when a class is matched by a class member in a program statement, predicated links in the statement will be interpretable in the data base on some particular LDP of that matched class member (see class use in Fig. 16).

To prevent anomalies arising, the syntax of class definitions includes certain restrictions. Each class LDP must be linked to an LDP of every class member LDP; if a shape is a class member, tokens of that shape may not be class members (see Fig. 15), since this could be redundant.

The system checks the syntax of class definitions at the users request or at compile time when it is abstracting from the class definition the information necessary to properly drive the interpreter. Class definitions may be edited and classes deleted like any other components of an AMBIT/G program.

The minor changes in the system have been occasioned by the desire to ease the burden of the user. One such change has been the modifying of the scope format to indicate the states of

-20-8508

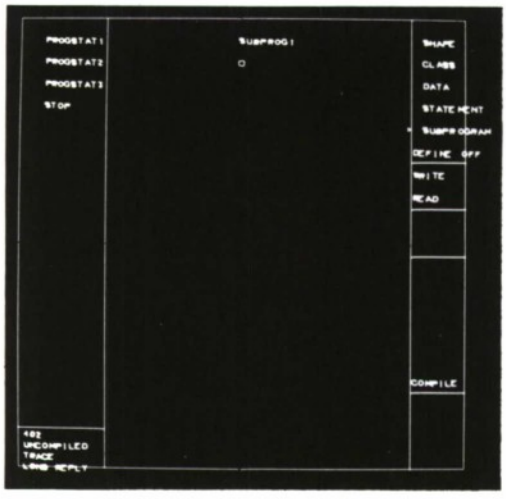


Fig. 11. Display when ready to define subprogram SUBPROG1. Notice at left, names of subroutines which can be called, and at top center, entry point to subprogram.

-20-8509

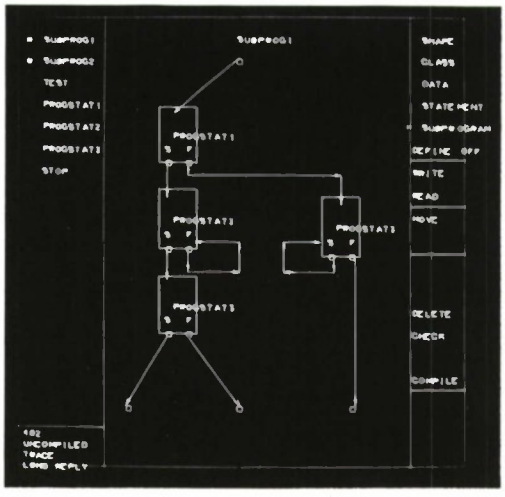


Fig. 12. Subprogram SUBPROG1 defined with three exit points, single calls to PROGSTAT1 and PROGSTAT2, and two calls to PROGSTAT3. Arrows indicate flow of control. Notice success (S) and failure (F) exits on all calls.

-20-8510

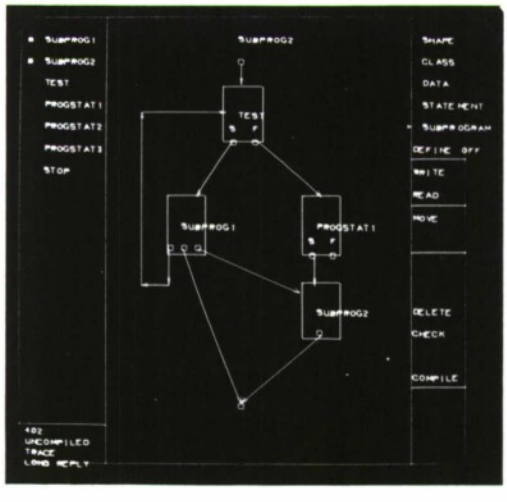


Fig. 13. Definition of subprogram SUBPROG2. Notice that calls to subroutines which are subprograms have varying numbers of exit points. Note also that this subprogram calls itself recursively.

-20-8511

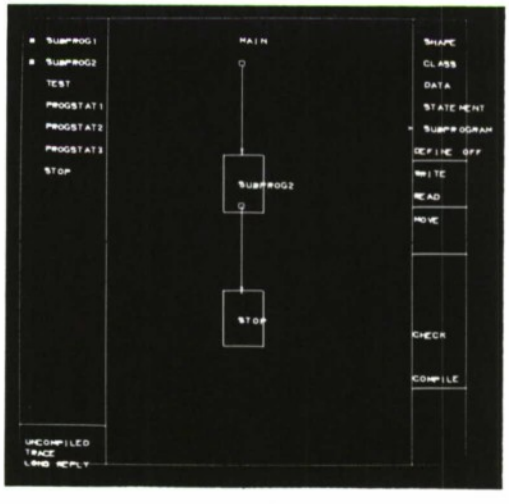


Fig. 14. Definition of main subprogram. It has no exit point and may not be called by subprograms. Notice reserved subprogram STOP.

the system and to provide only relevant light targets at any given time. The figures indicate these variations.

The save and restore commands have been augmented to allow not only a partially defined, but also a partially or totally interpreted, program to be permanently saved in the user's directory. He is thus much better protected against time-sharing system and machine failures.

Four reserved shapes have been added to the language (see Fig. 16). "σ" means "anything" and acts like a class with all defined shapes as members. "ϕ" means "nothing" and can be used to ask if a certain LDP is "linked to nothing," i. e., not yet linked, and to indicate that a certain link should be changed to "link to nothing," i. e., destroyed. "P" causes the interpreter to pause at that statement; "N" causes the name of the statement to be printed when that statement is encountered. This provides a selective hardcopy trace of the program.

The need to trace a program has been met in another way. While the program is being interpreted, a growing display of the statements executed and subprograms entered and exited can be shown. Since the interpreter runs very slowly, this growth is slow enough to allow the user to intelligently watch what is happening in the flow of control, and interrupt the interpreter if he should wish. A HELP light button is provided to allow such interaction with the program. This HELP light button is essential to terminate the interpretation prematurely in such circumstances as program loops. The use of the HELP button, breakpoints, and the "Pause" reserved shape all cause the system to enter a "partially run" state. At this time, and indeed whenever a program has been compiled successfully, the output mode may be entered. Many small improvements have been made to the output program to allow, for example, scaling the display and creation of a display with cycles in it. Care has been taken to make the display faithfully represent the data base, a fact which leads to some rather complex calculations when erasure of sections of the display takes place.

The availability of a character recognizer has been very helpful, and most of the commands to the system are now either light buttons or hand-drawn characters, the latter being changeable to suit individual preference.

System responses and error messages have been made graphical as much as possible, but the input of names has remained dependent upon the keyboard. The requests for such names are typed and come in long and short forms, intended for experienced and novice system users, respectively. A status switch may be set indicating which form the user desires. The status of this switch and of the trace switch is indicated in the lower left corner of the display along with the program state.

The use of a Tektronix 611 storage scope provides the system user with a subsidiary display which is used for looking at the statements and subprograms called in subroutines.

Current comment from users is providing some indicated directions for further improvements and/or experimentation with the system. A project is proposed to add some language forms for associating algebraic values with nodes in the data base and manipulating them with an imbedded algebraic language. The design of these additions has yet to be settled.

The work to date has been a good proving ground for various ideas in interactive graphical programming, primarily in the design of system responses and displays. This AMBIT/G system tries to unburden the user by providing an environment which implicitly or explicitly contains as much of what needs to be remembered as possible. Thus, a user can appeal to his

-20-8512

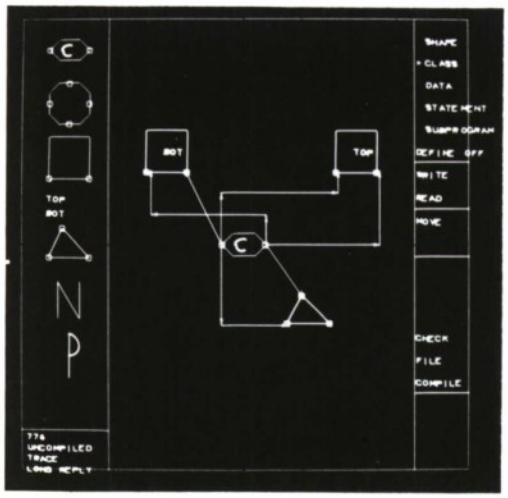


Fig. 15. A class definitian with three members in closs. Notice LDP correspondence indicotions far eoch closs member.

-20-8513

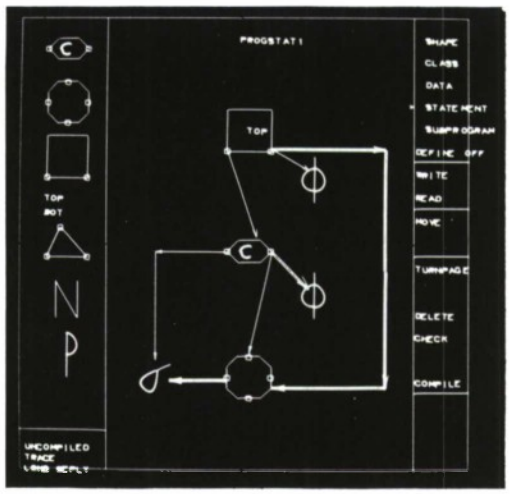


Fig. 16. Progrom stotement shawing use of o closs ond special reserved shapes.

-20-8514

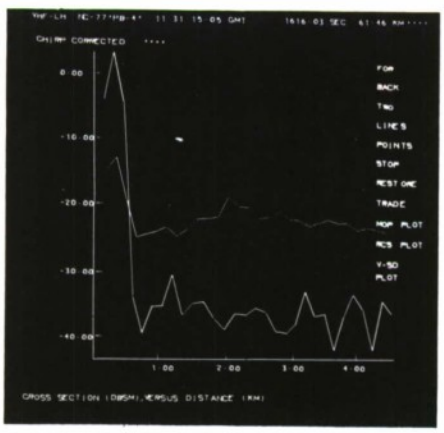


Fig. 17. Rodor trode plot showing cross-sectionol view af periadogram plats. Cammand light buttons ore in column ot right of picture.

-20-8515

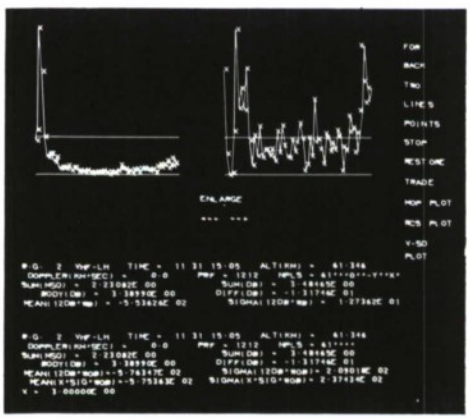
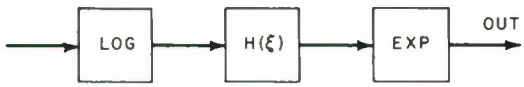


Fig. 18. Twa rodor periodograms (with ond without body data) far a single ronge gate.

environment to determine what his actions should be, rather than having to remember where he is.

C. Waveform Processing

The main research endeavor has focused on the cancelling of certain optical illusions through multiplicative filtering of visual stimuli. A simple model of the eye has been developed, and comparisons of this model with previously accepted models have been made. Cancellation of the spatial frequency response of the eye has been obtained assuming a multiplicative model. A functional specification of the frequency response of this model is given below.

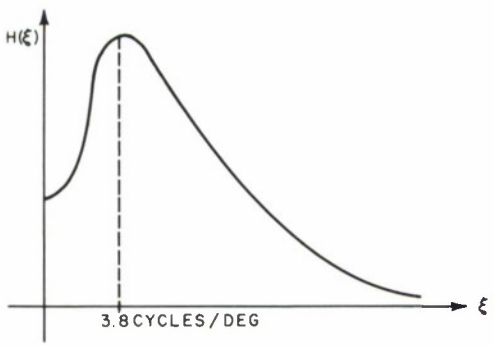


where:

ξ = spatial frequency

$$= \frac{A}{1 + 4\pi^2 B\xi^2} - \frac{C}{1 + 4\pi^2 D\xi^2}$$

A, B, C, D constants



D. RSP (Re-entry Systems Program)

The program demonstrating techniques for editing RSP data has reached a terminal state. Radar data from the Laboratory's IBM 360 can be transferred to TX-2. Program input consists of one or more data tapes comprised of a series of periodogram and trade plots. The plots can be displayed with the ability to select data values from the descriptive text corresponding to each plot to generate new plots. Hard copies of any or all of the plots can be obtained. Control and selection of the plots to be viewed is accomplished via tablet stylus input by the person editing data. The data exist in sequences of plots and these may be scanned either in a forward or reverse direction. Plots of associated data can be doubled up for comparison. Figures 17 and 18 show typical displays of the data.

One of the purposes of editing these data is to build another plot using data points selected from many of the raw data graphs. As the editor browses through the sequences of data, he can designate the pertinent values for inclusion in the plot being constructed. This resulting graph can then be viewed to ascertain its properties of interest.

E. Testing Terminal

The TX-2 testing terminal, TIC, is to be augmented with a small dedicated machine. The new terminal, called T^3, will connect to the TX-2 via a relatively low bandwidth link; the TX-2 will be used primarily for bulk storage of test results. T^3 will have a disk memory of sufficient size and speed to allow for large test programs and data files.

Because we feel it necessary that the design engineers and technicians do their own applications programming and interfacing of test equipment, we plan to implement an easy-to-learn and use, single-language operating system for the small machine. The language will be interpreted on T^3 in order to provide superior debugging aids and protection. Later, a compiler for the same language will be implemented on the TX-2. The user will first run and debug his programs interpretively; then, debugged subprograms for which faster execution speed is desired could be compiled.

F. Computer-Aided Logic Design

A library of programs, called LLL (Low-Level Logic), has been developed to aid in the LSI processor development. The foundation of this development is a graphics program to assist the logic designer in drawing, simulating, and documenting logic diagrams. Employing the tablet and scope, the designer can draw logic diagrams using standard gates (NAND, NOR, etc., including wired connections that perform a logic function, i. e., wired-or), delay elements, wires, and input and output pads. Elements of the diagram can be moved, deleted, and labeled with an arbitrary character string. A macro facility is available in which a logic diagram can be named, associated with a symbol, and then called by name for use in larger diagrams. Macros can be iterated to any depth.

A fundamental feature of this design program is that information about connectivity is not generated during the drawing and editing stages. This makes moving and deleting easy to handle (and hence fast) in the data structure. An operation called "acceptance" generates the connectivity information and graphically indicates those portions of the diagram that are inconsistent or drawn so that an unambiguous interpretation is impossible.

Once a logic diagram has been completely accepted, the user can write logic values on any set of leads, and all logic values implied by the written values will be computed and displayed. If the diagram contains delays or feedback, this simulation is done in increments of one delay time, where all delay elements are assumed equal. The most obvious use of this simulation is to verify that the circuit does what the designer intended. If mistakes are observed, the diagram can be unaccepted, edited, and reaccepted until the designer is satisfied.

At this point, a variety of programs designed to find test sets for the logic (provided it is combinational) can be called. Test sets, either minimal or near minimal in size, can be computed for the detection of single stuck-at-0 or stuck-at-1 faults. The complete test set for the location of an arbitrary single gate failure can be found. In addition, the tablet can be used to specify constraints on the diagram, and programs can be called to determine whether input vectors exist that satisfy the constraints. These constraints can be arbitrary combinations of logical values and sensitivity conditions.

A diagram and associated data structure, as it exists at any point in the procedure described above, can be saved for future reference. A facility is available for describing existing circuits

and inputting, via paper tape or typewriter, their description. The simulation and test generation programs can be used with typewriter control. All features described are fully operational and will shortly be evaluated in an actual design environment.

Figures 19 through 22 are presented to illustrate LLL. An accumulating register is constructed from flip-flop and full adder macros.

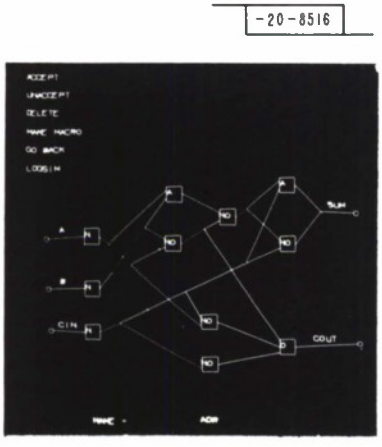


Fig. 19. Example af a macra definitian, a full adder given the name ADR. There is ane virtual gate, the wired-or, whase autput lead is SUM.

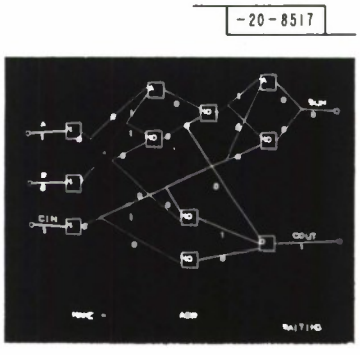


Fig. 20. ADR simulated far input {A,B,CIN} = {1,0,1} which gives autput {SUM, COUT} = {0,1}. Nate that leads which are user-specified (farcing values) are brighter.

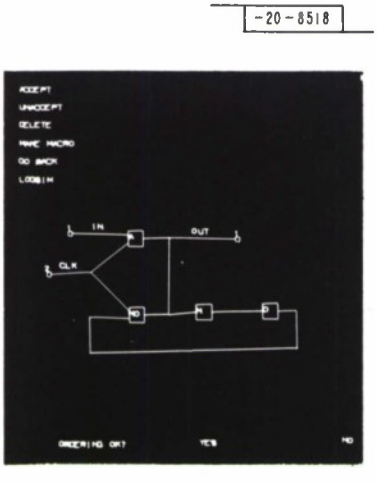


Fig. 21. A flip-flap macra definition which will be named FF. At this paint, LLL is requesting user ta accept ar rearder input and autput leads. Nate numbers aver pads. All feedback loops must be braken by delay elements (D).

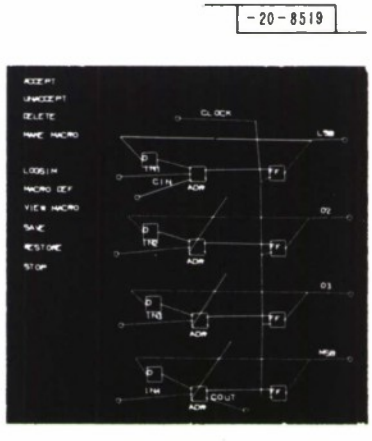


Fig. 22. Macras FF and ADR have been used ta canstruct a 4-bit accumulating register. Carry aut af bit i is cannected ta carry input af bit i + 1. Outputs {LSB,02,03,MSB} will be added ta inputs {IN1,IN2,IN3,IN4, and CIN} when clack is true, ance far each simulatian cycle.

G. Conic Generator and Display Consoles

A redesigned conic generator was installed in the TX-2 display. With the new system, line and curve quality at 20 MHz is better than it formerly was at 1 MHz. Overall performance of the display is now limited by the scopes themselves. Deflection delay and bandwidth limitations result in short gaps at the ends of lines drawn at high rates (> 5 MHz). A scope circuit modification has been designed and successfully tested on one console and will soon be made on all consoles. At that time, all consoles will have displays usable at a 20-MHz drawing rate.

Tektronix 611 storage scopes are being added to TX-2 consoles, thus making two display tubes available to each user. The storage scope can be used in refresh only, store only, or a combination of the two modes and, like the standard scope, is handled by the display executive portion of APEX but in a different manner. The approach taken for the standard scopes was to make the display executive handle the details of structuring and buffering a display file. Previous reports have outlined the mechanism and display structure employed. For the storage scope, the opposite approach has been taken. The user has full responsibility for his data and must concern himself with hardware idiosyncrasies in the display generator. The executive does the actual transmitting of the information to the scope and also guarantees that no matter what the user does, he cannot affect any displays except his own.

The storage mode does not need the concept of a structured display since no target hit feedback is possible. Consequently, a simple block transfer of output data is possible, and this can be handled with the TX-2 channel hardware.

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Lincoln Laboratory, M.I.T.	Unclassified
	2b. GROUP: None

3. REPORT TITLE

Semiannual Technical Summary Report to the Advanced Research Projects Agency on Graphics

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Semiannual Technical Summary Report (1 June through 30 November 1968)

5. AUTHOR(S) (Last name, first name, initial)

Raffel, Jack I.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
30 November 1968	28	3

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
AF 19(628)-5167	Semiannual Technical Summary for 30 November 1968
b. PROJECT NO. ARPA Order 691	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	ESD-TR-68-350

10. AVAILABILITY/LIMITATION NOTICES

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Advanced Research Projects Agency, Department of Defense

13. ABSTRACT

A BCPL compiler is being transferred to the TX-2 from Project MAC. A microprogramming assembler has been written, and microcodes can be generated for a prototype processor under construction. The operation of the microcodes and the instruction sets they define can be simulated. An easily trainable public character recognition service has been made available to the TX-2 users community, and it can be called directly from LEAP programs.

The mask design facility has been improved and used to generate artwork for several test circuits. In addition, development of an IBM 360 capability has continued to the point of real use by Laboratory personnel. The AMBIT/G programming system has been refined in that drawn programs are themselves AMBIT/G data graphs. The Waveform Processing and RSP projects have been completed.

The TIC testing terminal is being augmented with a small stand-alone computer. The logic design and simulation programs have been further developed to permit the input of data, observation of values, and generation of test sequences for the circuit under design.

The conic display generator has had a major speed improvement, and storage scopes are being added to all TX-2 consoles.

14. KEY WORDS

graphical communication
TX-2
time-sharing
man-machine
display systems
programming languages